Finding Objects through Stochastic Shortest Path Problems

Felipe W. Trevizan
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA, USA
fwt@cs.cmu.edu

Manuela M. Veloso
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, USA
mmv@cs.cmu.edu

ABSTRACT

This paper presents a novel formulation for the problem of finding objects in a known environment while minimizing the search cost. Our approach consists in formalizing this class of problems as Stochastic Shortest Path (SSP) problems, a decision-theoretic framework for probabilistic environments. The obtained problems are solved by using off-the-shelf domain-independent probabilistic planners. The advantages of this approach includes: (i) a well defined optimization problem in which the probability of finding the object is maximized while minimizing the cost of searching for the object; and (ii) being able to take advantage, without any modifications to our model, of any (future) technique in the field of domain-independent probabilistic planners, such as better algorithms and better heuristics. We also contribute by empirically comparing three probabilistic planners algorithms, namely FF-Replan, UCT and SSiPP, using our proposed class of problems.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Plan execution, formation, and generation

General Terms

Algorithms

Keywords

Planning, replanning, Stochastic Shortest Path Problems

1. INTRODUCTION

The problem of an autonomous agent moving in an environment to find objects while minimizing the search cost is ubiquitous in the real world, e.g., a taxi driver looking for passengers and minimizing the usage of gas, a software agent finding information about a product in the web while minimizing the bandwidth usage, a service robot bringing objects to users minimizing distance traversed, and a robot collecting rocks for experiments while minimizing its battery consumption. In all these problems, we assume that the agent does not know where the exact objects are, and has some probabilistic model of the location of the objects. In this work, we present how to model this class of problems as probabilistic planning problems.

For this paper, our concrete motivation is the mobile service robot that moves in a building to find an object, e.g., *coffee*, and to deliver it to a location, e.g., office #171. We assume that the agent is given a map of the environment and that the object can be in more than one location. Also, we consider that the probability of the object being at a location type, e.g., offices, is given. Such prior distribution can be designed by an expert or automatically obtained, for example by querying the web (e.g., [16]). In this paper, we focus on the problem of finding the object, since the delivery problem can be cast as finding an object that is deterministically present only in the delivery location.

Given the probabilistic nature of the finding object problems, one of the contributions of this work is the formulation of the considered class of problems as Stochastic Shortest Path problems (SSPs) [3]. Therefore, the objective function of the obtained problems is to maximize the probability of finding the object while minimizing the cost of searching for the object in the environment.

We show how to encode the obtained SSPs in a standard probabilistic planning language, which allows any off-the-shelf domain-independent probabilistic planners to solve them. The advantages of this approach include: (i) the usage of a well understood optimization problem, namely the SSPs; and (ii) being able to take advantage, without any modifications to our model, of future improvements in the field of domain-independent probabilistic planners, such as better algorithms and better heuristics. We also contribute by empirically comparing three probabilistic planners algorithms, namely FF-Replan [20], UCT [11] and SSiPP [17], using our proposed class of problems for different object prior distributions.

The remainder of this paper is organized as follows: Section 2 reviews the related work. Section 3 formally defines SSPs. Section 4 presents our model for finding an object in the environment. Section 5 reviews domain-independent probabilistic planning algorithms and Section 6 empirically evaluates them using our modeled problems. Section 7 concludes the paper.

2. RELATED WORK

The problem of finding an object using a camera is explored by [1]. Their approach consists in decoupling the route planning problem from the object finding problem and the latter is modeled as an *MDP over the belief space*. In

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.*

their proposed model for object finding, a state is a belief over the relational descriptions for the object location in the current location, e.g., "book on table in living room". Due to the intractability of belief space MDPs, they use greedy search over a finite horizon to find an object search strategy.

Mapping objects while moving through a known environment is the task considered by [18]. They assume a deterministic motion model for the robot and a probabilistic model for the object sensors, in which the object being detected depends on the robot location. Sampling over a fixed action horizon is employed to find a path that minimizes the traveled distance and an expected loss over false positive and false negative detections of objects.

A different approach for finding objects in an environment is given by [12]. Their approach does not assume a prior distribution for the location of the desired object and this prior is inferred using data extracted from the web about the co-occurrence between the desired object and objects which the prior is known. A deterministic motion model is assumed and the proposed objective function is the expected length of the plan to reach the object, where the expectation is over the a posteriori probability of finding the object given the known objects observed so far. Breadth-first search with additional constraints to avoid undesired paths, e.g., traverse each location at most twice, over a finite horizon is applied to solve the obtained problems.

Another approach that uses the web to obtain the probability distribution of an object being in a given location type is ObjectEval [16]. ObjectEval also computes a sequence of locations to be visited in order to find the desired object. This is done by maximizing an *ad hoc* utility function based on the probability of finding the object, the distance travelled, the cost to obtain the object and the feedback about the object, i.e., if the object is known to (not) be in an specific location. Beam search is used for finding an exploration route. Therefore ObjectEval is not complete, i.e., it might fail to find the object before all locations are explored.

Our work differs from all the above approaches by considering the route planning problem and the object finding problem as a single probabilistic planning problem. Because of this unified formulation, we don't need to assume a deterministic motion model and we can use domain-independent probabilistic planners to compute a *policy* that maximizes the probability of finding the object while minimizing the cost of searching for the object. Moreover, if an optimal (complete) probabilistic planner is used, then our approach is optimal (complete).

3. STOCHASTIC SHORTEST PATH PROBLEMS

A Stochastic Shortest Path Problem (SSP) [3] is defined as the tuple $\mathbb{S} = \langle \mathbf{S}, s_0, \mathbf{G}, \mathbf{A}, P, C \rangle$. An SSP describes a control problem where $\mathbf{S}$ is the finite set of states of the system, $s_0 \in \mathbf{S}$ is the initial state and $\mathbf{G} \subseteq \mathbf{S}$ is the set of (absorving) goal states. Actions $a \in \mathbf{A}$ control transitions from one state s to another state s' and these transitions happen with probability $P(s'|s, a)$. When a state s' is reached after action a is applied on state s, the agent pays the cost $C(s, a, s') \in (0, +\infty)$. An example of SSP is depicted in Figure 1.

Markov Decision Processes (MDPs) [14] and SSPs are similar models and they differ in their *horizon*, i.e., how many actions the agent is allowed to execute. While MDPs have

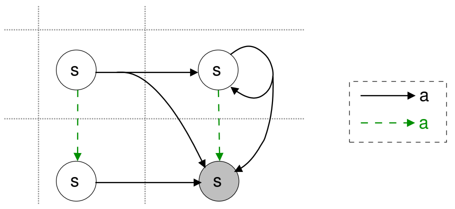

Figure 1: Example of an SSP. The initial state is s_0, the goal set $\mathbf{G} = \{s_3\}$ and $C(s, a, s') = 1$ $\forall s \in \mathbf{S}, a \in \mathbf{A}, s' \in \mathbf{S}$. This SSP can be represented as a factored SSP with two binary state variables, x and y, such that the state $\langle x, y \rangle$ equals the state s_i for $i = x + 2y$.

either a finite or infinite horizon, SSPs have an *indefinite horizon*, that is, the agent is allowed to perform as many actions as needed in order reach a goal state $s_G \in \mathbf{G}$. Therefore, if a goal state is reachable from every state $s \in \mathbf{S}$ for an SSP $\mathbb{S}$, then a finite and unknown number of action will be applied until a goal state is reached in $\mathbb{S}$. Since the length of this action sequence is finite, the total action cost accumulated is also finite. Both finite and infinite horizon MDPs can be represented as SSPs [2].

A solution for an SSP is a policy π, i.e., a mapping from $\mathbf{S}$ to $\mathbf{A}$. An optimal policy π^* for an SSP is any policy that always reaches a goal state when followed from s_0 and also minimizes the expected accumulated cost to do so. The optimal expected cost to reach a goal state from a state s, represented by $V^*(s)$, is the unique solution to the fixed point equations defined by (1) for all $s \in \mathbf{S}$. Once V^* is known, any optimal policy π^* can be extracted from V^* by substituting the operator min by argmin in equation (1).

$$V(s) = \begin{cases} 0, & \text{if } s \in \mathbf{G} \\ \min_{a \in \mathbf{A}} \sum_{s' \in \mathbf{S}} P(s'|s, a) \left[C(s, a, s') + V(s') \right], & \text{otherwise} \end{cases} \tag{1}$$

The enumerative specification of $\mathbb{S}$ can be burdensome for large problems, specially the encoding of $P(\cdot|\cdot, a)$ as a matrix $\mathbf{S} \times \mathbf{S}$ for each action a. To compactly represent $\mathbb{S}$, we can use *state variables*, i.e., properties whose value changes over the time [6]. In this factored representation, a set $\mathbf{F} = \{f_1, \cdots, f_k\}$ of states variables is given and $\mathcal{D}_i$ is the domain of $f_i \in \mathbf{F}$. The state space $\mathbf{S}$ is the cross product $\times_{i=1}^{|\mathcal{F}|} \mathcal{D}_i$ and a state $s \in \mathbf{S}$ is the tuple $\langle v_0, \cdots, v_{|\mathcal{F}|} \rangle$ where $v_i \in \mathcal{D}_i$. For example, the SSP in Figure 1 can be factored using two binary state variables, x and y, such that state $\langle x, y \rangle$ equals the state s_i for $i = x + 2y$. For the rest of this paper, we assume the domain of each state variable $f \in \mathbf{F}$ to be binary, thus $|\mathbf{S}| = 2^{|\mathbf{F}|}$.

Another benefit of using state variables is a compact representation of $P(\cdot|\cdot, a)$ [6]. Consider action a_0 of the SSP depicted in Figure 1. The enumerative representation of $P(\cdot|\cdot, a_0)$ is a 4-by-4 stochastic matrix, which is encoded with $4 \times 3 = 12$ numbers. For this example, a factored representation is $P(\langle x', y' \rangle | \langle x, y \rangle, a_0) = P(x'|x, a_0) \times P(y'|y, a_0)$ where $P(x' = 1|x = 0, a_0) = 0.25$, $P(x' = 1|x = 1, a_0) = 1$ and $P(y' = 1|y = 0, a_0) = P(y' = 1|y = 1, a_0) = 1$, which can be encoded with only 4 numbers.

The Probabilistic Planning Domain Description Language (PPDDL) [19] is a standard language to represent factored SSPs that is used in the international probabilistic planning

```
(:action a0
 :effect (and (y) (prob 0.25 (x) 0.75 (not (x))))
)
(:action a1
 :precondition (not (x))
 :effect (x)
)
```

Figure 2: PPDDL code for the actions in the SSP depicted in Figure 1. Note that only action a1 has a precondition.

competitions (IPPC) [5,7,21]. PPDDL syntax is based on LISP and an action a consists of a precondition, that is, a formula over the state variables characterizing the states in which a is applicable, and an effect. The effect describes how the states variables change when a is applied. Any state variable not explicitly modified by a remains unchanged after executing a (frame assumption). Figure 2 contains the PPDDL representation of the actions a_0 and a_1 of the SSP represented in Figure 1.

PPDDL also features *predicates* and *action schemas*. These extensions use the concept of domain variables, i.e., class of finite objects. A predicate is mapping from a value assignment of one or more domain variables to a state variables. For instance, we can model a graph $G = \langle N, E\rangle$ by using a domain variable called NODE in which its domain is N and edges of the graph as the predicate edge(i, j) where i and j are domain variables of the type NODE; in this case, each possible instantiation of edge(i, j) represents one binary state variable. Therefore, if the planning problem defines three objects of the type NODE, namely n_1, n_2, n_3, then six state variables are instantiated representing the edges $(n_1, n_2),(n_1, n_3),(n_2, n_1),\ldots,(n_3, n_2)$. Similarly to predicates, action schemas map value assignment of one or more domain variables to an action.

4. USING SSPS TO FIND OBJECTS

In this section we present the PPDDL encoding for finding an object in a building. For this encoding, we use one domain variable, LOCATION, representing the locations the agent is allowed to visit and the following predicates defined over locations:

- connected(l_1, l_2): true when the agent can move from location l_1 to l_2;
- at(l): to represent the agent's current location;
- objAt(l): to denote that an instance of the object being search is at l;
- searched(l): to indicate that l has already being searched;
- and a set of predicates to denote the type of each location, e.g., isOffice(l) for office locations and isKitchen(l) for kitchens.

Also, we use the state variable hasObject to indicate that the agent has the desired object.

For each location type t, we use the binary random variable X_t to denote if the object is at the locations of type t and we assume that a prior probability $\bar{P}(X_t)$ is given. Note that $\sum_t \bar{P}(X_t = \text{true})$ is not required to sum up to 1. This feature is used for representing scenarios such as an object that can be found deterministically in more than one location type or an object that has a low probability to be found in any location type. To simplify notation, we denote $\bar{P}(X_t = \text{true})$ as p_t for every location type t.

```
(:action Search
 :parameters (?l - location)
 :precondition (and (at ?l) (not (searched ?l)))
 :effect (and
    (searched ?l)
    (when (isBathroom ?l) (prob 0.08 (objAt ?l)))
    (when (isKitchen ?l)  (prob 0.18 (objAt ?l)))
    (when (isOffice ?l)   (prob 0.02 (objAt ?l)))
    (when (isPrinterR ?l) (prob 0.72 (objAt ?l))))
)
```

Figure 3: PPDDL code for the action Search(l). For this action the prior used for the object being at a location l is 8%, 18%, 2% and 72% if l is, respectively, a bathroom, a kitchen, an office or a printer room.

We model the object finding through a pair of action schemas, Search and PickUp. The action Search(l), depicted in Figure 3, has the precondition that the agent is at location l and l has not been searched before. Its effect is searched(l), i.e., to mark l as searched, and, with probability p_t, where t is the location type of l, the object is found. With probability $1 - p_t$, the object is not found at l. Since searched(l) is true after the execution of Search(l), the agent cannot search the same location l more than once. This is enforced because $(1 - p_t)^k \to 0$ as $k \to \infty$ for $p_t > 0$, i.e., if the agent were allowed to search the same location enough times it would always find the object there.

```
(:action PickUp
 :parameters (?l - location)
 :precondition (and (at ?l) (objAt ?l))
 :effect (and
   (not (objAt ?loc))
   (hasObject))
)
```

Figure 4: PPDDL code for the action PickUp(l).

The action PickUp(l), depicted in Figure 4, represents the agent obtaining the object at location l if the object is there. This action can be easily extend to encompass more general scenarios, e.g., a robotic agent with grippers that can fail and the object might not be always obtained or a symbiotic autonomous agent that might ask people for help to manipulation the object [15]. Such extensions can be modeled by converting PickUp(l) into a probabilistic action or a chain of probabilistic actions.

We use the action schema Move to model the agent moving in the map represented by the predicate connected(l_1, l_2). The action Move(l_1, l_2) is probabilistic and with probability p the agent moves from l_1 to l_2 and with probability $1 - p$ the agent stays at l_1. For all the examples and experiments in the remaining of this paper we use $p = 0.9$.

Initially, the value of the state variable hasObject is false and the goal of the agent is to reach any state in which hasObject is true. For easy of presentation, we define the cost of all actions to be 1, i.e., $C(s, a, s') = 1\ \forall s \in \mathbf{S}$, $a \in \mathbf{A}$, $s' \in \mathbf{S}$. Therefore the average cost of reaching the goal equals the average number of actions applied by the agent.

To illustrate our model, consider the map presented in Figure 5.(a). In this map, the agent is at position 0 and there are two hallways that can be explored: (i) the right

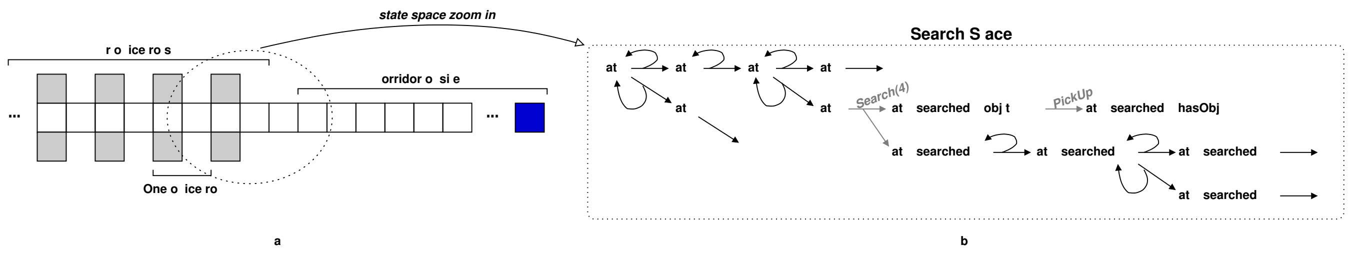

Figure 5: (a) Example of map representing a building. The agent is initially at location 0. Gray cells represent offices, the dark blue cell represents the kitchen and white cells represent the hallways. (b) Visualization of the initial portion of the search space for the map on (a). Arrows depict actions: arrows with self-loop represent the action `Move`, gray arrows represent either `Search` or `PickUp`. Due to the closed-world assumption, any state variable not presented in (b) is considered false. State $\langle$`at`(4),`searched`(4),`hasObj`$\rangle$ is a goal state.

hallway of size k in which the last location is a kitchen; and (ii) the left hallway with $2r$ offices. Notice that Figure 5.(a) represents only the map of the environment and not the search space. A fraction of the search space is depicted on Figure 5.(b).

In order to show the effects of each parameter in the solution of the problem, consider the policies π_j, for $j \in \{0, \dots, r\}$, in which the agent explores the first j offices rows, then explores the kitchen and finally the remaining $r-j$ offices row. For all π_j, the exploration stops once the object is found. For instance, if $p_{\text{office}} = 1$, then the only policy that explores the kitchen is π_0 since no office is explored before the kitchen, and all other policies stop exploring after the first office is visited.

Figure 6 shows the average cost of following the policies π_j from the location 0 in map from Figure 5.(a). Each plot of Figure 6 varies either k, r, p_{kitchen} or p_{office} while fixing the other parameters to $k = 10$, $r = 10$, $p_{\text{kitchen}} = 0.9$, $p_{\text{office}} = 0.1$. The top right plot shows that the average cost of π_j is exponential in r. This is due to the exponential effect of the probability of not finding the object in a sequence of i offices, i.e., $(1 - p_{\text{office}})^i$. Also, the optimal policy, i.e., the lowest π_j at any point of the plots, is either exploring the kitchen first (π_0) or all the offices first (π_r) for this example.

5. PROBABILISTIC PLANNERS

One approach to solve SSPs is to directly find the V^* using dynamic programming to iteratively update the set of equations (1). This approach, known as value iteration, is optimal [3] and computes a closed policy, i.e., an universal mapping function from every state in $s \in \mathbf{S}$ to the optimal action $\pi^*(s)$. Closed policies are extremely powerful as their execution never "fails", i.e., every probabilistic outcome of the actions is taking into account, and the planner is never re-invoked. Unfortunately the computation of closed policies is prohibitive in complexity as problems scale up. For instance, in the example depicted in Figure 5, a closed policy encompasses all the 2^{2r+1} possible combination of explored locations, what is infeasible for large values of r.

The efficiency of value iteration based probabilistic planners can be improved by combining asynchronous updates and heuristic search (e.g., Labeled RTDP [4]). Although these techniques allow planners to compute compact closed policies, in the worst case these policies are still linear in the size of the state space, which itself can be exponential in the number of state variables or goals.

Another approach to solve SSPs is by replanning. Replanners do not invest the computational effort to generate a closed policy, and instead, compute a partial policy, i.e., a mapping from a subset of $\mathbf{S}$ to actions. Since a partial policy π does not address all the probabilistic possible reachable states from s_0, during its execution in the environment, a state s in which $\pi(s)$ is not defined can be reached. If and when such state s is reached, the replanner is re-invoked to compute a new partial policy starting from s.

A simple and powerful approach for replanning is determinization, i.e., to relax the probabilistic problem into a deterministic problem D and use a deterministic planner to solve D. The winner of the first International Probabilistic Planning Competition (IPPC) [21], FF-Replan [20], is a replanner based on the *all-outcomes* determinization that uses the deterministic planner FF [9] to solve D. The all-outcomes determinization is obtained when $\mathbf{A}$ is replaced by the set $\mathbf{A}' = \{s \to s' | \exists a \in \mathbf{A} \text{ s.t. } P(s'|s, a) > 0\}$ of deterministic actions. The drawback of determinizations is being oblivious to the probability of each outcome of actions and their correlation. To illustrate this drawback, consider the example in Figure 5, for $k = 5$, $r = 1000$, $p_{\text{kitchen}} = 1.0$ and $p_{\text{office}} = 0.001$, and its all-outcomes determinization D. This deterministic problem D has two actions to represent the probabilistic action `Search`: (i) a_1 that always finds the object in the current location; and (i) a_2 that never finds the object. Therefore the deterministic planner solves D by moving the agent to the location 4 (Figure 5), i.e., the closest location where the object can be found with positive probability, and applying action a_1 to deterministically find the object. However, in the original problem, the object is found in location 4 with probability 0.001 and if the object is not there, this approach will visit all the other $2r - 1$ offices until the object is found. Notice that the optimal solution in this case is to explore the kitchen first since the object is found there with probability 1.

Sampling, another replanning approach, is employed by the Upper Confidence bound for Trees (UCT) [11]. UCT is an approximation of the t-look-ahead search [13] obtained by using sparse sampling techniques. Formally, UCT iteratively builds a policy tree by expanding the best node according to a biased version of (1) to ensure that promising actions are sampled more often. Notice that UCT, as t-look-ahead search, builds a policy tree, i.e., a policy free of loop, since the horizon of the problem is relaxed from indefinite to finite of size t. While UCT does not require the parameter t, it is governed by two other parameters: w the number of samples per decision step and c the weight of the bias term

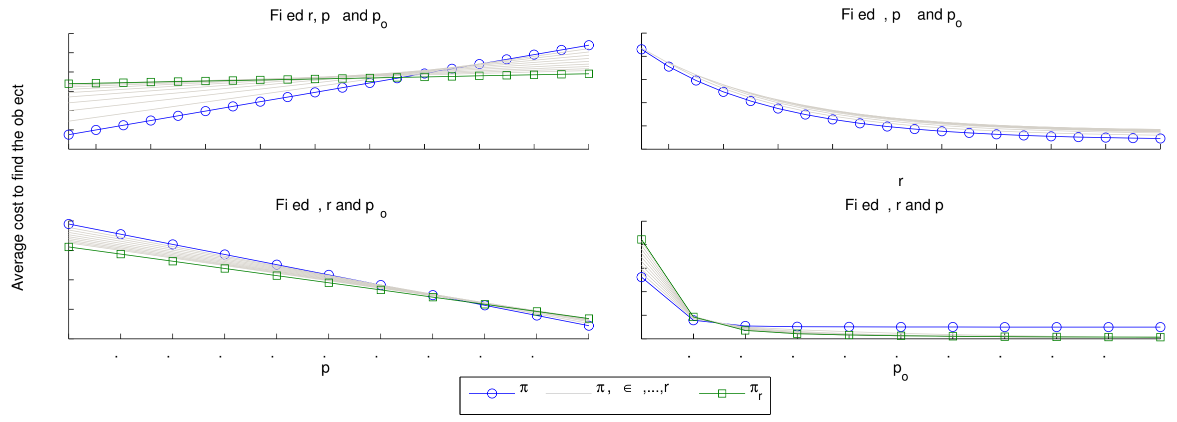

Figure 6: Average cost of the policies π_j in the map depicted in Figure 5.(a) for $k = 10$, $r = 10$, $p_{\textbf{kitchen}} = 0.9$ and $p_{\textbf{office}} = 0.1$. In each plot, one of the four parameters is varied in the x-axis. In all plots, the best policy (bottom curve) is either π_0 (explore the kitchen and then the offices) or π_r (explore the offices and then the kitchen). In the top right plot, the policy π_r varies as a function of r, the x-axis, and is depicted in gray for clarity.

for choosing actions. UCT is the base of PROST [10], the winner of IPPC 2011 [8].

An orthogonal direction from all other approaches mentioned so far is the *short-sighted search* employed by SSiPP [17]. SSiPP consists in incrementally generating and solving subproblems of the original SSP. These subproblems, known as (s, t)-short-sighted SSP, are SSPs in which: (i) every state has positive probability of being reached from s using at most t actions; and (ii) the states only reachable using more than t action are approximated by an heuristic that estimates their cost to reach a goal state. An off-the-shelf optimal probabilistic planner is used for solving a given (s, t)-short-sighted SSP and the obtained policy $\hat{\pi}$ can be followed from s for at least t actions without replanning. An important difference between UCT (and t-look-ahead) and SSiPP is that the former relaxes the original problem by limiting its indefinite horizon to a fixed finite horizon while SSiPP approximates the original problem by pruning the state space without changing its horizon. Therefore, SSiPP can perfectly represent loop of actions of size up to t, such as the self-loop action `Move`. Alternatively, UCT only reason about a fixed number of *failures* of the action `Move`, i.e., the effect of staying in the same location with probability 0.1 is considered to happen only a finite number of times in a row.

One advantage of UCT and SSiPP over FF-Replan is that they do not over simplify the neighborhood of current state in their search space. Therefore, there are values for the parameters of UCT and SSiPP in which they can find better solutions than FF-Replan while still being computationally feasible. For instance, consider the example in Figure 5, for the parameters $k = 5$, $r = 1000$, $p_{\text{kitchen}} = 1.0$ and $p_{\text{office}} = 0.001$, as before. For large values of w, at least one sample of UCT will reach the kitchen and the policy of exploring the kitchen first (π_0) is selected. If $t \geq 7$, SSiPP returns π_0 independently of the heuristic used.

6. EXPERIMENTS

We present five different experiments, each of them for a different object over the same map. The objects considered in the experiments are: coffee, cup, papers, pen and toner. The prior distribution for the object location is obtained using ObjectEval [16] and shown in Table 1. We consider that the object is never in the hallways, i.e., $p_{\text{hallway}} = 0$.

Object	**Location**			
	Bathroom	Kitchen	Office	Printer Room
coffee	0.08	**0.72**	0.18	0.02
cup	**0.42**	0.36	0.12	0.10
papers	0.00	0.13	**0.70**	0.17
pen	0.15	0.23	**0.35**	0.27
toner	0.05	0.02	0.06	**0.87**

Table 1: Prior probability obtained by ObjectEval [16] for the object being in a given location type. The mode of each prior is shown in bold.

For all the experiments, we consider the map depicted in Figure 7. The graph representing this map contains 126 edges and 121 nodes, i.e., locations: 2 bathrooms, 2 kitchens, 59 offices, 1 printer room and 57 segments of hallway. Since there is no location in which any of the considered objects can be found with probability 1, then, with positive probability, the object might not be found after visiting all locations. This probability is approximately 5×10^{-7}, 6×10^{-5}, 9×10^{-32}, 2×10^{-12} and 3×10^{-3} for *coffee*, *cup*, *papers*, *pen* and *toner*, respectively. The simulations in which this low probability event happens are ignored and rerun.

The planners considered in the experiments are FF-Replan, UCT and SSiPP. For the latter two, we use the FF-heuristic h_{ff}: for a given state s, $h_{\text{ff}}(s)$ equals the number of actions in the plan returned by FF using s as initial state and the all-outcomes determinization. For SSiPP, we use Labeled RTDP as the underlying optimal probabilistic planner [4,17] and $t \in \{2, 4, 6, \cdots, 20\}$. We consider 12 different parametrizations for UCT obtained by using the bias parameter $c \in \{1, 2, 4, 8\}$ and the number of samples per decision $w \in \{10, 100, 1000\}$. The experiments are conducted in a Linux machine with 4 cores running at 3.07GHz and planners have a 3Gb memory cut-off and 10 minutes cpu-time cut-off.

			UCT $w = 1000$			SSiPP					
	l_0	FF-Replan	$c = 2$	$c = 4$	$c = 8$	$t = 10$	$t = 12$	$t = 14$	$t = 16$	$t = 18$	$t = 20$
coffee	1	17.9 ±3	18.8 ±7	19.9 ±7	23.1 ±9	19.9 ±4	18.5 ±3	19.3 ±4	20.8 ±3	20.7 ±3	17.2 ±3
	2	19.4 ±3	18.0 ±7	23.3 ±8	22.2 ±9	14.2 ±3	13.3 ±2	14.1 ±3	13.6 ±3	12.9 ±2	13.0 ±2
	3	13.7 ±5	12.3 ±5	13.4 ±8	11.4 ±5	9.7 ±2	10.5 ±3	8.6 ±2	8.1 ±2	8.8 ±2	8.0 ±2
	4	18.5 ±4	17.2 ±9	18.1 ±10	16.2 ±4	12.2 ±3	13.1 ±3	12.2 ±3	11.9 ±2	11.3 ±2	12.1 ±2
	5	14.5 ±3	14.9 ±4	15.1 ±4	14.7 ±10	14.9 ±2	15.4 ±3	16.7 ±4	17.2 ±3	13.6 ±2	14.2 ±2
	6	21.6 ±3	22.7 ±8	24.4 ±10	23.2 ±11	19.1 ±4	21.7 ±4	19.5 ±4	19.1 ±4	18.7 ±2	18.0 ±4
	7	21.3 ±4	37.6 ±9	36.3 ±11	34.1 ±12	25.8 ±4	20.5 ±4	21.2 ±4	20.0 ±3	20.8 ±4	20.8 ±3
	8	17.3 ±4	27.7 ±9	22.1 ±9	31.5 ±8	13.9 ±3	13.0 ±2	14.5 ±4	14.4 ±3	12.9 ±3	15.0 ±3
	9	15.2 ±4	14.0 ±6	17.8 ±7	18.7 ±6	11.6 ±3	10.0 ±3	13.0 ±3	11.5 ±3	10.0 ±3	11.8 ±3
	10	20.5 ±5	17.3 ±8	25.9 ±9	23.4 ±7	20.8 ±4	16.4 ±3	15.1 ±3	15.9 ±3	14.1 ±3	14.3 ±3
cup	1	28.4 ±5	38.9 ±9	34.7 ±8	31.9 ±8	35.4 ±4	30.8 ±5	29.8 ±6	30.1 ±7	29.3 ±7	27.6 ±5
	2	33.5 ±6	31.8 ±9	27.0 ±9	26.6 ±8	30.7 ±6	25.6 ±5	26.4 ±6	27.7 ±6	23.3 ±4	24.0 ±5
	3	27.6 ±4	30.8 ±9	33.8 ±10	29.4 ±10	25.5 ±6	26.2 ±7	23.9 ±5	19.5 ±5	17.7 ±3	19.1 ±4
	4	34.0 ±6	41.6 ±10	48.6 ±9	35.5 ±9	25.9 ±5	23.0 ±5	23.8 ±5	22.9 ±4	24.5 ±5	24.5 ±5
	5	30.5 ±5	35.5 ±8	36.8 ±8	42.3 ±9	29.6 ±6	25.0 ±5	28.4 ±6	25.2 ±4	27.3 ±5	24.3 ±4
	6	30.3 ±6	41.5 ±9	37.1 ±9	33.7 ±8	34.3 ±6	28.7 ±5	24.1 ±5	26.2 ±5	20.7 ±3	21.1 ±4
	7	28.1 ±5	30.6 ±8	35.8 ±8	33.3 ±8	34.5 ±6	23.8 ±5	22.9 ±4	29.2 ±7	21.6 ±5	23.2 ±6
	8	35.4 ±7	20.7 ±9	24.5 ±10	21.9 ±11	24.1 ±6	21.9 ±4	19.9 ±6	20.9 ±5	18.0 ±4	20.8 ±5
	9	35.9 ±8	29.3 ±10	25.6 ±8	26.4 ±9	29.3 ±7	19.3 ±6	19.9 ±6	15.9 ±4	15.5 ±5	15.3 ±3
	10	31.4 ±6	37.4 ±10	23.7 ±10	27.6 ±8	23.3 ±4	27.6 ±6	22.4 ±4	24.0 ±5	20.9 ±4	20.8 ±4
papers	1	3.3 ±1	3.2 ±1	3.9 ±1	3.9 ±2	3.2 ±0	3.6 ±1	3.2 ±0	3.8 ±1	3.3 ±0	3.6 ±1
	2	3.7 ±1	3.7 ±1	3.1 ±1	4.4 ±1	4.0 ±1	3.7 ±1	4.2 ±1	3.5 ±1	3.8 ±1	3.4 ±1
	3	4.4 ±1	4.9 ±1	4.4 ±1	4.8 ±1	3.7 ±1	3.5 ±1	3.8 ±1	3.8 ±1	3.5 ±1	3.6 ±1
	4	4.4 ±1	4.3 ±1	4.7 ±1	4.9 ±3	3.6 ±1	3.7 ±1	3.5 ±1	3.5 ±1	3.6 ±1	3.7 ±1
	5	3.5 ±1	3.4 ±1	3.9 ±1	3.3 ±1	3.7 ±1	3.9 ±1	3.4 ±1	3.9 ±1	3.5 ±1	3.4 ±1
	6	3.6 ±1	3.7 ±1	3.9 ±1	3.8 ±1	3.5 ±1	3.5 ±1	3.9 ±1	3.6 ±1	3.4 ±1	3.6 ±1
	7	5.9 ±1	6.4 ±1	6.2 ±1	6.0 ±1	6.0 ±1	6.1 ±1	6.0 ±1	5.8 ±1	6.2 ±1	5.8 ±1
	8	4.7 ±1	3.9 ±1	3.5 ±1	3.8 ±1	4.4 ±1	3.5 ±1	3.9 ±1	3.6 ±1	3.6 ±1	3.7 ±1
	9	4.8 ±1	3.5 ±1	3.7 ±1	4.0 ±1	4.0 ±1	3.5 ±1	3.9 ±1	3.8 ±1	3.8 ±1	3.8 ±1
	10	3.4 ±0	3.3 ±1	4.1 ±2	3.5 ±1	3.2 ±1	3.3 ±0	3.5 ±1	3.4 ±1	3.7 ±1	3.5 ±1
pen	1	9.4 ±2	9.1 ±3	8.7 ±3	9.3 ±4	9.0 ±2	10.2 ±2	8.7 ±2	8.5 ±2	9.1 ±2	8.4 ±1
	2	8.8 ±2	8.9 ±4	9.0 ±2	8.7 ±3	9.8 ±2	9.2 ±2	9.8 ±2	8.5 ±1	8.9 ±2	8.9 ±2
	3	8.5 ±1	10.8 ±3	10.8 ±3	12.0 ±3	9.5 ±2	8.2 ±2	9.5 ±2	8.9 ±2	8.7 ±2	7.8 ±1
	4	8.2 ±2	9.6 ±3	10.4 ±3	9.1 ±3	9.2 ±2	8.3 ±2	9.0 ±2	8.7 ±3	9.0 ±2	8.5 ±2
	5	8.7 ±2	9.6 ±3	8.6 ±2	9.7 ±5	9.6 ±1	9.9 ±2	8.8 ±2	9.0 ±2	9.4 ±2	9.1 ±2
	6	11.1 ±3	11.0 ±3	11.7 ±2	10.8 ±3	11.0 ±2	10.7 ±1	10.6 ±2	10.0 ±2	10.1 ±2	10.0 ±2
	7	10.9 ±2	11.7 ±3	11.9 ±3	11.4 ±4	11.4 ±2	11.1 ±2	11.2 ±2	11.3 ±2	11.2 ±2	11.5 ±2
	8	10.7 ±2	10.4 ±3	10.9 ±2	10.5 ±3	10.1 ±2	11.8 ±2	8.6 ±2	10.8 ±2	10.4 ±2	10.2 ±2
	9	11.3 ±2	10.4 ±3	10.6 ±3	10.9 ±4	10.2 ±2	10.9 ±2	10.8 ±2	10.9 ±2	10.0 ±2	10.9 ±2
	10	9.7 ±2	9.3 ±2	9.9 ±2	9.7 ±2	9.4 ±2	9.8 ±2	9.5 ±2	9.6 ±2	9.9 ±2	9.5 ±2
toner	1	54.1 ±9	43.2 ±10	41.9 ±11	41.3 ±11	42.8 ±7	29.5 ±7	27.2 ±5	37.9 ±7	27.1 ±6	27.9 ±6
	2	56.8 ±9	41.9 ±10	45.7 ±12	40.3 ±11	41.5 ±5	19.0 ±5	18.3 ±5	18.7 ±5	18.5 ±6	18.3 ±6
	3	50.1 ±9	56.6 ±12	55.3 ±11	53.1 ±13	38.5 ±5	33.1 ±6	25.3 ±6	22.4 ±4	23.4 ±9	21.2 ±5
	4	61.3 ±9	59.3 ±10	58.0 ±12	42.2 ±11	30.2 ±9	20.7 ±6	20.5 ±6	19.1 ±7	21.3 ±7	19.3 ±7
	5	39.3 ±6	38.9 ±10	31.5 ±10	36.5 ±12	30.2 ±7	31.8 ±8	23.9 ±5	23.2 ±6	25.0 ±7	23.6 ±7
	6	53.3 ±6	37.5 ±11	29.8 ±7	23.1 ±6	18.6 ±6	19.6 ±4	19.0 ±5	18.9 ±6	18.4 ±4	18.6 ±6
	7	45.5 ±7	26.4 ±10	20.7 ±8	21.2 ±7	18.3 ±5	17.9 ±5	18.0 ±6	18.4 ±7	17.6 ±7	17.9 ±5
	8	33.9 ±8	21.5 ±10	19.8 ±12	18.7 ±9	23.4 ±10	19.7 ±9	18.8 ±6	16.7 ±8	16.2 ±8	17.1 ±7
	9	36.8 ±8	29.9 ±10	25.9 ±10	23.6 ±9	18.5 ±8	17.6 ±6	18.8 ±7	18.3 ±9	16.6 ±6	16.2 ±5
	10	54.5 ±8	31.5 ±9	29.5 ±7	27.6 ±10	27.8 ±6	25.1 ±6	23.0 ±6	24.1 ±7	22.6 ±7	22.1 ±6

Table 2: Average and 95% confidence interval of the number of actions applied to find the given object starting at location l_0 (Figure 7). The gray cells show the best performance for the given problem, i.e., the combinations of objects and initial locations represented by each line of the table.

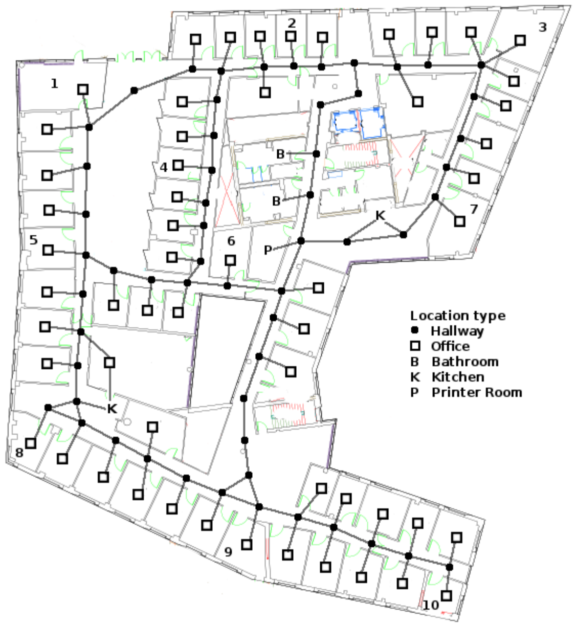

Figure 7: Floor plant considered for the experiments. The embedded graph represents the map given to the planners. The initial location for the experiments are represented by the numbers 1,. . . ,10.

The methodology for the experiments is as follows: each planner solves the same problem, i.e., find a giving object from a particular initial location, 100 times. Learning is not allowed, that is, SSiPP and UCT cannot use the bounds obtained in previous solutions of the same problem to improve their performance. The average number of actions performed in each problem is presented in Table 2. Due to space limitations, we present only 3 parametrizations of UCT and 6 parametrizations of SSiPP. All the omitted parametrizations perform consistently worst than the presented ones.

Overall, SSiPP performs better than the other planners in 55 problems out of 60 (approximately 92%) while the FF-Replan and UCT are the best planner in 3 and 4 problems respectively. Another clear trend is that as t increases for SSiPP, the better is its performance. This is expected since the behavior of SSiPP approaches the behavior of its underlying optimal planner, in this case LRTDP, as t increases. However, this improvement in performance is obtained by increasing the search space and consequently the running time of SSiPP. This trade-off between performance and computational time is shown in Figure 8 where the run time of the planners is presented.

Looking at specific objects and their priors, we can categorize the objects into: abundant, uniformly distributed and rare. An example of abundant object in the experiments is *papers* since its prior is 0.7 for office locations and offices represent 48% of the locations. Thus, the probability of not finding *papers* is the lowest between all the object considered: approximately 9×10^{-32}. Therefore, finding objects of this category is not a hard task and optimistic approaches, such as FF-Replan, perform well. This effect is illustrated by the results in third block of Table 2 in which the 95% confidence interval of every planner considerably overlaps. A similar phenomenon happens with uniformly distributed objects, i.e., objects in which their prior is *close* to an uniform distribution, represented in the experiments by *pen*.

A more challenging problem is posed by rare objects, i.e., objects in which their prior probability is concentrated in very few locations. In this experiment, *coffee*, *cup* and *toner* can be seen as rare objects. As expected, FF-Replan performs poorly for rare objects and extra reasoning is necessary in order to efficiently explore the state space. For instance, consider finding the object *cup* starting at position 7. Both a kitchen and an office are 3 steps away from position 7. In the all-outcomes determinization used by FF-Replan, the planner will have access to a deterministic action that always finds *cup* in the office and in the kitchen, therefore FF-Replan will randomly break the tie between exploring the kitchen and the neighboring office from position 7. If the office is explored, then FF-Replan will explore all the other offices in the hallway between positions 7 and 3 because they will be the closest locations not explored yet. Since the prior for *cup* is 0.12 for offices, a better policy is to explore the kitchen (prior 0.36) and then the two bathrooms (prior 0.42) that are at distance 4 and 5 of the kitchen.

The improvement in performance over FF-Replan is remarkable for the rare object *toner*, which its prior has mode 0.87 in one single location, the printer room. For these problems, both UCT and SSiPP present better performances than FF-Replan and the average number of actions applied by SSiPP, for $t \geq 14$, is approximately half of the average number of actions applied by FF-Replan.

7. CONCLUSION

In this paper, we presented how to solve the problem of a software or robotic agent moving in a known environment in order to find an object using domain-independent probabilistic planners. This is done by modeling the problem as a Stochastic Shortest Path problem (SSP) encoded in a standard probabilistic planning language and using a off-the-shelf probabilistic planner to solve it. The advantages this approach are two folds: (i) the usage of SSPs defines a well understood optimization problem in which the probability of finding the object is maximized while minimizing the cost of searching for the object; and (ii) improvements in the field of domain-independent probabilistic planners, such as better algorithms and better heuristics, can be directed employed in our model without modifications. Our approach also contributes by providing a new series of real-world inspired probabilistic planning problems.

We empirically compared three different replanning techniques to solve the proposed problems: determinizations (FF-Replan), sampling (UCT) and short-sighted planning (SSiPP). The experiments showed that the simpler and optimistic approach used by FF-Replan suffices if the object can be found in most locations with high probability or nearly uniform across over all locations. Alternatively, if the probability of finding the object is concentrated in few locations, then SSiPP outperforms the other approaches and, for some parametrizations, SSiPP executes on average less than half of the actions executed by FF-Replan to find the desired object.

Interesting extensions to explore as future work include: (i) develop an heuristic for choosing a suitable probabilistic planner for solving the current problem; and (ii) improve the prior of the given object by learning through the experiences of the agent.

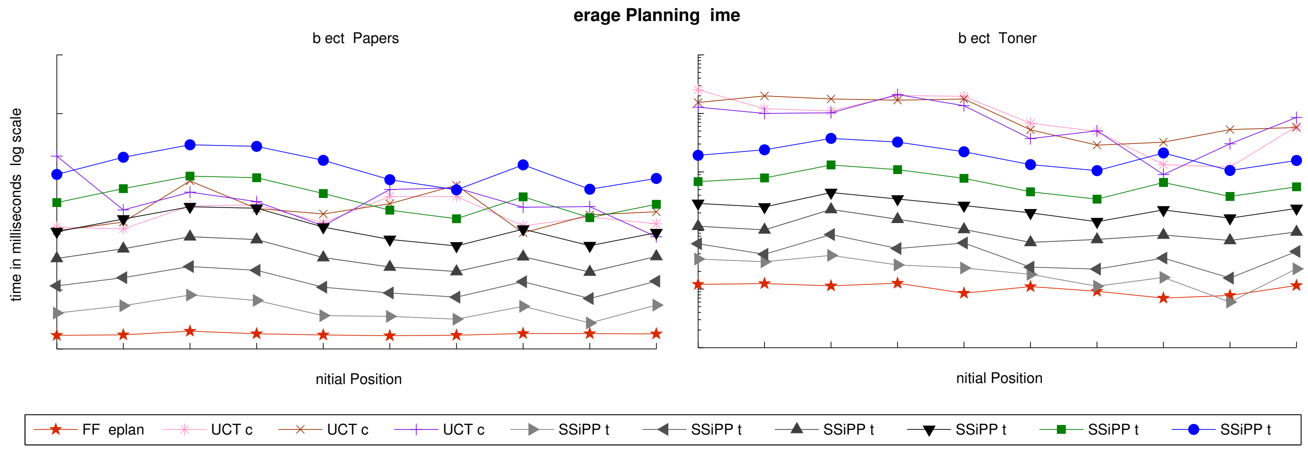

Figure 8: Average run time for the planners to find the objects *papers* and *toner* in milliseconds (y-axis in log-scale). Error bars omitted for clarity. The plot for the other objects follows a similar pattern, with SSiPP for $t = 12$ always faster than UCT planners for $w = 1000$.

8. REFERENCES

[1] A. Aydemir, K. Sjoo, J. Folkesson, A. Pronobis, and P. Jensfelt. Search in the real world: Active visual object search based on spatial relations. In *Int. Conf. on Robotics and Automation (ICRA)*, 2011.

[2] D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[3] D. Bertsekas and J. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

[4] B. Bonet and H. Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proc. of the 13th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2003.

[5] B. Bonet and R. Givan. 2th Int. Probabilistic Planning Competition (IPPC-ICAPS). `http://www.ldc.usb.ve/~bonet/ipc5/` (accessed on Oct 10, 2012), 2007.

[6] C. Boutilier, T. Dean, and S. Hanks. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[7] D. Bryce and O. Buffet. 6th Int. Planning Competition: Uncertainty Track. In *3rd Int. Probabilistic Planning Competition (IPPC-ICAPS)*, 2008.

[8] A. J. Coles, A. Coles, A. García Olaya, S. Jiménez, C. Linares López, S. Sanner, and S. Yoon. A survey of the seventh int. planning competition. *AI Magazine*, 33(1):83–88, 2012.

[9] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14(1):253–302, 2001.

[10] T. Keller and P. Eyerich. Probabilistic planning based on uct. In *Proc. of 22nd Int. Joint Conf. on Automated Planning and Scheduling (ICAPS)*, 2012.

[11] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo Planning. In *Proc. of the European Conf. on Machine Learning (ECML)*, 2006.

[12] T. Kollar and N. Roy. Utilizing object-object and object-scene context when planning to find things. In *Int. Conf. on Robotics and Automation (ICRA)*, 2009.

[13] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Menlo Park, California, 1985.

[14] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[15] S. Rosenthal, J. Biswas, and M. Veloso. An effective personal mobile robot agent through symbiotic human-robot interaction. In *Proc. of the 9th Int. Conf. on Autonomous Agents and Multiagent Systems*, 2010.

[16] M. Samadi, T. Kollar, and M. Veloso. Using the Web to Interactively Learn to Find Objects. In *Proc. of the 26th Conf. on Artificial Intelligence (AAAI), Toronto, Canada*, 2012.

[17] F. W. Trevizan and M. M. Veloso. Short-sighted stochastic shortest path problems. In *Proc. of the 22th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2012.

[18] J. Velez, G. Hemann, A. Huang, I. Posner, and N. Roy. Planning to perceive: Exploiting mobility for robust object detection. In *Proc. of the 21st Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2011.

[19] H. L. S. Yones and M. L. Littman. PPDDL 1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Mellon University, 2004.

[20] S. Yoon, A. Fern, and R. Givan. FF-Replan: A baseline for probabilistic planning. In *Proc. of the 17th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2007.

[21] H. Younes, M. Littman, D. Weissman, and J. Asmuth. The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research*, 24(1):851–887, 2005.